WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences

Search Results -

Terms	Documents
l9 and (tablespace or table space or table-space)	21

Database: US Patents Full-Text Database; US Pre-Grant Publication Full-Text Database; JPO Abstracts Database; EPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Refine Search: Clear

Search History

Today's Date: 9/28/2001

DB Name	Query	Hit Count	Set Name
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l9 and (tablespace or table space or table-space)	21	L10
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l8 and database	21	L9
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	table near recovery	171	L8
USPT	5926816.pn.	1	L7
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l5 and extract	26	L6
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l3 and rows	86	L5
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l3 and row with extraction	0	L4
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l2 and (tablespace or table space or table-space)	315	L3
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	l1 and table with recovery	315	L2
USPT,PGPB,JPAB,EPAB,DWPI,TDBD	database	83517	L1

L3: Entry 41 of 315 File: USPT Feb 13, 2001

DOCUMENT-IDENTIFIER: US 6189010 B1
TITLE: Method for repairing constraint violations in a database management system

ABPL:
In response to a constraint violation in a row of a database table, an output file is generated including the characteristics of the table containing the row in error as well as an SQL UPDATE statement for the row. The SQL UPDATE statement includes the column values in the row which can be corrected by the user, the user modified SQL UPDATE statement being subsequently executed to repair the constraint violation.

BSPR:
The present invention relates to database management systems, and particularly to a method for repairing constraint violations in a database management system.

BSPR:
A well known database software program is DATABASE 2 (DB2) database software distributed by IBM Corporation. As is known in the art, DB2 operates as a subsystem in a computer system operating under the IBM MVS operating system software. In a DB2 environment, user data resides in DB2 tables which are in tablespaces. A tablespace is, for example, a portion of storage space in a direct access storage device (DASD) such as a disk drive. For exemplary purposes, illustrated below is an order_entry table that would be stored in a tablespace. The order_entry table contains columns: customer_number; product_code; order_number; buyer_name; and ship_to_zip.

BSPR:
The index key can be used to generate an index for the table which facilitates subsequent searches for particular data in the table. For example, the Order_Entry table would have three indexes (e.g., one for each index key), each index being stored in an indexspace. Similar to a tablespace, an indexspace is, for example, a designated portion of a DASD. Thus, if a user was looking for rows that contain a particular buyer name in the Order Entry table, the database management system could query the buyer index for the table to identify all occurrences of the buyer name without reading the entire table to locate the rows.

BSPR:
DB2 administrators analyze performance characteristics for application programs that access a database table in an attempt to find the optimum index structure for fast access to the database table. The values to be used as an index must be carefully selected because each index results in overhead for the database system. For example, each transaction in a database table, such as an add or delete, requires that each index for the table also be updated. Thus, it is desirable that the number of indexes for a table be minimized to enhance the performance of application programs. The values to be used as an index for a database table are selected based on, for example, data accessed most frequently by users of the table, generally on-line transaction users. Index keys generally are not based on foreign keys, as foreign keys are used primarily for validation purposes (e.g., constraint enforcement).

BSPR:
As is known in the art, each table in a database may be either a parent table, a child table or both. A child table is related to a parent table via the foreign key value or values contained in columns of the child table. For example, a foreign key value can appear multiple times in a child table (e.g., multiple rows in a child table can have the same foreign key, such as the customer_number and product_code entries in the order_entry table) but each foreign key must be associated with a unique key in a parent table of the child table.

BSPR:
Referential integrity ensures that every foreign key value is valid (e.g., has a corresponding primary key in a parent table). Thus, referential integrity (RI) means that a value in the column of a row in the table is valid when this value also exists in an index of another table. A row should not be in a table if it violates a constraint.

As the order_entry table illustrated above has two foreign keys, it has a RI constraint on customer_number and product_code. As is known in the art, when a user of a DB2 database management system creates a table, the user also defines the constraints for the table (e.g., the user can define the relational integrity criteria). Illustrated below are an exemplary product table and an exemplary customer table (e.g., the parent tables for the foreign keys in the order_entry table).

BSPR:
The customer table illustrated below shows four rows, although this table could also have thousands of rows for all of the customers of a company. The customer table has, for example, an index based on the column customer_number, which values are illustrated in ascending order. The values in the column customer_number are each unique since there is only one customer_number assigned to each customer name and thus a customer_number would not be included in this table more than once. Accordingly, an index for the customer table would include the key value (e.g., the value of the column customer_number) and a RID. The customer index would also reside in a DB2 indexspace.

BSPR:
Conventional database management systems, such as DB2, provide the user with the ability to identify specific conditions that a row must meet before it can be added to a table. These conditions are referred to as "constraints" because they constrain the values that a row may include. Constraints include, for example, check constraints and referential integrity constraints. Check constraints include, for example, qualifying criteria for a particular value, such as a zip code value (e.g., the ship_to_zip value in the Order_Entry table) being in the range of 00000 to 99999. As discussed above, referential integrity constraints ensure that a value in a row of a table is valid when the value also exists in an index of another table.

BSPR:
Constraint enforcement can be performed prior to loading of data into a database table or after data has already been loaded into a database table. An example of performing constraint enforcement prior to loading data into a database table is provided in co-pending application Ser. no. 09/058,754 filed Apr. 10, 1998, owned by the Assignee of the present application and which is hereby expressly incorporated by reference. If constraint enforcement is performed after loading data into a database table, for example as part of a recovery operation following a hardware of software failure, the constraint enforcement is generally performed by a CHECK utility, such as CHECK DATA by IBM Corp., CHECK PLUS by BMC Software and FASTCHECK by Platinum technology, inc.

BSPR:
Conventional CHECK utilities ensure that data in the table do not violate any constraints that have been established for the table. Constraints can be established at the time the table is generated. For example, constraints can be defined when the table is originally created in the database system and are stored in the DB2 catalog, which can be subsequently queried by a CHECK utility to identify the constraint information.

BSPR:
To perform constraint enforcement, a conventional CHECK utility would, for example, be initialized and identify any applicable constraints for the table to be checked by reading the DB2 catalog, as is known in the art. The CHECK utility would, for example, then read each row of the database table and check for check constraint violations and/or referential integrity constraint violations.

BSPR:
Constraints may be violated for numerous reasons, not all of which require that the row containing the error be deleted. For example, a user might want to correct the error rather than delete the row. An option with some CHECK utilities, such as CHECK PLUS by BMC Software and FASTCHECK by Platinum technology is that if a constraint violation is identified, and thus a row of the database table contains an error, the CHECK utility will produce a DB2 SQL DELETE statement for each row containing a constraint violation. These SQL DELETE statements will be written out to a sequential file which the customer can then use to selectively delete particular rows that are in violation of the constraint(s). For example, in the Order_Entry table illustrated above, a data entry error in the customer_number would not render the order invalid but rather may reflect an error that can be corrected, thus allowing continued processing of the order (e.g., it may be more desirable to fix the error than delete the row).

BSPR:
The SQL DELETE statements generated by conventional CHECK utilities are based on the foreign key value that violates the constraint. For example, when the SQL DELETE statement is generated by the CHECK utility, the statement uses the key value causing the constraint violation. For example, if the customer number 99999 was read by a CHECK utility while checking the Order_Entry table illustrated above and the customer number 99999 violated a referential integrity constraint (e.g., it was not in a parent index

for the parent customer table, then the CHECK utility would generate a SQL DELETE statement using the foreign key value that violated the constraint (i.e., customer number=99999).

BSPR:
Thus, when the SQL DELETE statement is generated using a key value that is not indexed for the table and is executed, the entire database table must be read for each DELETE statement to determine if the foreign key value identified in the SQL DELETE statement is contained in any row of the database table. As mentioned previously, generally foreign key values are not used as an index for a database table and thus no index values are available for use by the DELETE statements utilizing a foreign key value to facilitate deletion of rows containing errors. Therefore, as each DELETE statement generated by the CHECK utility deletes any row that matches the foreign key that is in error, and if the database table contains millions of rows, each of the millions of rows would have to be read for each executed DELETE statement. If even only two rows contain the key value used in the DELETE statement, all of the rows of the table would have to be read. Thus, significant processing time can be consumed as part of the constraint enforcement process due to the need to read an entire database table multiple times.

BSPR:
When a constraint violation is identified, for example due to a DASD failure where a database table is recovered and the recovery process has induced some errors, a user would probably not want to delete the row in error. Instead, the user would want to fix the row, particularly where the data was valid before the failure. Similarly, a LOAD operation without constraint enforcement could be used to load new rows containing customer orders into an Order Entry table. If any constraint violations are identified in the newly loaded data, the user would want to correct the errors, not delete new orders. Conventional CHECK utilities do not, however, provide a tool to correct the rows in error other than deletion of the rows.

BSPR:
For example, the generation of SQL DELETE statements by conventional CHECK utilities do not provide a mechanism to facilitate correction of selected rows containing a constraint violation. The generation of the DELETE statements only provides the option of deleting the row. Correction of the row requires that the user generate a script (e.g., SQL code) to make any desired corrections for each row in error. For example, to correct a row identified as containing a constraint violation, the user of the database system must develop a corrective action plan. Usually, the user will manually code and test SQL statements to fix the rows in error. If there are, for example, 1,000 rows in error, the user would have to produce 1,000 SQL statements to correct the rows as each row requires its own SQL statement. The generation of numerous SQL statement is often plagued by errors introduced during the correction process. Further, most database management system users have numerous DB2 tables to maintain, for example 1,000 DB2 tables, each table having multiple columns of varying data types that may contain an error, thus further complicating the row correction process. Therefore, there is a need to improve the error correction process for rows containing a constraint violation.

BSPR:
In addition, the mechanism used by conventional CHECK utilities used to identify a row containing a constraint violation, typically the row identification (RID), has limited value to a user. The user would prefer to know the data value in error, for example a customer number or an order number which is a value that is meaningful to the user, instead of a page number and row number that can be used by DB2. In contrast to the limited and DB2-oriented information provided by conventional CHECK utilities, the database user wants as much as information as possible rather than merely the location of the row-the user is more interested in the entire contents of the row including the columns in error.

BSPR:
According to an embodiment of the present invention, in response to a constraint violation in a row of a database table, an output file is generated to facilitate updating the rows containing a constraint violation. The output file includes, for example, the characteristics of the table containing the row in error as well as an SQL UPDATE statement for each row in error. The SQL UPDATE statement is automatically generated and includes, for example, the values of the columns in the row, any of which can be corrected by the user. The modified SQL UPDATE statement can be subsequently executed to implement the desired corrections in the database table.

BSPL:
While the above Order_Entry table shows four rows, the table could have millions of rows for all the orders of a company, for example 4 million rows. The order_entry table also has, for example, three index keys and two foreign keys. An index key is an identifier for a particular row of a table while a foreign key also identifies a row but is also used for referential integrity as described below. For example, in the order_entry

table, one index key could b[redacted]ased on Order_Number, another i[redacted]x key based on buyer_name and a third index key based on ship_to_zip. As is known in the art, an index key for a particular table indicates a row identification (RID) and a selected value for the row (e.g., the index key value).

BSPL:
The product table show five rows, although the table could have thousands of rows for all of the different products of a company. The product table has, for example, an index based on the column product_code, which values are illustrated in ascending order. The values in the column product_code are each unique since there is only one product code assigned to each product and thus in this table, a product code would not be included more than once. Accordingly, an index for the product table would include the key value (e.g., the stored value in the product_code column) and a RID. The product table index would reside in a DB2 indexspace.

BSPL:
As shown by the above tables, all of the rows in the Order_Entry table are valid (e.g., there are no referential integrity constraint violations) because the foreign key values in the column product_code of the Order_Entry table also exist in the product table and the values in the column customer_number of the Order_Entry table also exist in the customer table.

BSTL:
Order_Entry Table customer_number product_code order_number buyer_name ship_to_zip 1111111111 0010 1234500001 John Doe 60606 1111111111 0040 1234500002 Jane Doe 70707 3333333333 0020 1234500003 Bill Smith 90909 2222222222 0030 1234500004 Fred Smith 80808

BSTL:
Product Table product_code product_description retail_price 00010 laptop pc 1000.00 00020 desktop pc 1100.00 00030 office pc 1200.00 00040 lan pc 3500.00 00050 home pc 999.99

BSTL:
Customer Table customer_number buyer_name customer_address 1111111111 John Doe State A 2222222222 Fred Smith State B 3333333333 Bill Smith State C 4444444444 Steve Jones State D

DRPR:
FIG. 1 illustrates an indexspace and a tablespace to be checked for constraint violations according to an exemplary embodiment of the present invention.

DRPR:
FIG. 3 is an exemplary database table according to an embodiment of the present invention.

DEPR:
FIG. 1 illustrates a tablespace 110 including a database table 120 and an indexspace 130 including an index 140. FIG. 3 illustrates a more detailed representation of table 120. Similar to the example Order_Entry table used earlier, table 120 includes columns 311-315 and rows 321-325 for an exemplary Order Entry table. The intersection of each column and row of table 120 contains an entry, which may be a unique or non-unique value. For example, column 311 contains the customer_number entries, which may be non-unique values as the same customer may have multiple orders pending in the Order_Entry table 120. Column 313 of table 120 contains order_number entries, each of which is a unique value as each order placed by a customer is assigned a unique number. Also as shown in FIG. 3, column 312 in table 120 contains product_code entries, column 314 contains buyer_name entries and column 315 contains ship_to_zip entries, all of which are non-unique values.

DEPR:
FIG. 2 illustrates a more detailed representation of index 140. Index 140 illustrates an index for table 120 on the order_number column 313 of table 120. Accordingly, index 140 includes two columns 211 and 212 containing, respectively, a row identification (RID) and associated index value, in this case the order_number entries for rows 322-325 of table 120. The order_number column is selected as an index for the Order_Entry table 120 because, for example, it is not a foreign key value for table 120 and the order numbers are each unique values. The ship_to_zip or buyer_name columns could also be used as index values for table 120. Index 140 is defined, for example, after the database table has been created and a database administrator has conducted some conventional performance analysis on the table, for example evaluating the most commonly accessed values of the database table by on-line users, to determine which column of the table should be selected for the index 140.

DEPR:

FIG. 4 illustrates an exemplary flowchart for a method of enforcing constraints according to an embodiment of the present invention. In step 410, a CHECK utility is initialized to enforce constraints on a database table, such as table 120 illustrated in

DEPR:
FIG. 2. As is known in the art CHECK utilities are generally initialized each time a tablespace is to be checked. The CHECK utility could include, for example, CHECKDATA by IBM Corp., CHECK PLUS by BMC Software or FASTCHECK by Platinum technology, inc. In step 420, the CHECK utility reads a row of a database table, data already having been loaded into the database table. In step 430, check constraint enforcement is performed. Check constraint enforcement can be performed in any conventional manner. In step 440, it is determined if there are any check constraint violations. If a check constraint violation is identified in step 440, then error processing according to an embodiment of the present invention occurs in step 450. For example, a printed report of the constraint violation is generated and a SQL DELETE statement is generated so that the row containing the constraint violation can be deleted. According to the present invention, the SQL DELETE statement is generated by the CHECK utility utilizing an index value associated with the row containing the constraint violation, which may differs from the foreign key value causing the constraint violation. In another embodiment of the present invention, the CHECK utility can also check if a DELETE statement has already been generated for the row due to a previous check constraint violation, thus avoiding generation of duplicative DELETE statements.

DEPR:
If no check constraint violation is identified in step 440, then in step 460, referential integrity constraint enforcement is performed. Referential integrity constraint enforcement can be performed in any conventional manner. If a referential integrity violation is detected in step 470, then error processing according to an embodiment of the present invention occurs in step 480. For example, a printed report of the constraint violation is generated and a SQL DELETE statement is generated utilizing the index value associated with the row. A check can be made to verify that a DELETE statement for the row has not yet already been generated, for example, due to a previously identified CHECK constraint violation. For example, a flag can be set in the CHECK utility when a DELETE statement is generated for a row. The status of the flag (e.g., set or not set) can be verified by the CHECK utility for each row prior to generating a DELETE statements for the row, thereby avoiding duplicative generation of DELETE statement for a particular row. After error processing in step 480 is completed, or if no referential integrity violation is identified in step 470, the process returns to read the next row in the database table at step 420, this looping continuing until each row in the database table has been reviewed for constraint violations.

DEPR:
FIG. 5 illustrates exemplary initialization processing of a CHECK utility according to an embodiment of the present invention, such as performed in step 410 of FIG. 4. In step 510, a user of the CHECK utility provides a name of a tablespace which is to be subject to constraint enforcement. For example, the user can input the name of the tablespace via an I/O device such as a keyboard to the computer system operating the database management system and the CHECK utility. In step 520, the CHECK utility identifies the database table located in the tablespace identified by the user, for example by reading the DB2 catalog. Usually, there is only one database table in a tablespace.

DEPR:
In step 530, the CHECK utility identifies the columns in the database table so that, for example, when an index is identified for the table the CHECK utility can identify the column name for the index value and locate the corresponding column in the database table to retrieve the index value to be used in generating the SQL DELETE statement according to an embodiment of the present invention. Performing this step during initialization precludes the need to identify the appropriate column in the database table each time a SQL DELETE statement is generated. Also, during the initialization process, the CHECK utility can flag the column(s) in the database table that contain index values and subsequently locate the flagged columns if there is a constraint violation to obtain the column name and value to be used in generating the DELETE statement.

DEPR:
In step 540, the CHECK utility identifies any check constraints that may apply to the table. Similarly, in step 550, the CHECK utility identifies any referential integrity constraints that apply to the database table. In step 560, the CHECK utility identifies the index or indexes that have been defined for the table. If only one index is defined for the table, then that index is identified by the CHECK utility. If more than one index is defined for the table, then the CHECK utility would, for example, select the index based on unique values, which can be determined via the DB2 catalog (e.g., the uniquerule data contained in the DB2 catalog). If no unique index value exists, then the

CHECK utility could use, for example, the first non-unique index. In step 570, the CHECK utility identifies the column names for the index defined for the database table and identified by the CHECK utility.

DEPR:
Applicable check constraints can be identified by, for example, the CHECK utility reading the DB2 catalog (e.g., the SYSIBM.SYSCHECKS table in the catalog). The check constraints that apply are defined, for example, when the table is created in the database system and are stored in the DB2 catalog. The following is an example of how a conventional CHECK utility would read the SYSCHECKS table in a DB2 catalog.

DEPR:
Referential integrity constraints applicable to the table can be identified by a conventional CHECK utility by, for example, the CHECK utility reading the DB2 catalog (e.g., the SYSIBM.SYSRELS table of the DB2 catalog). The referential integrity constraints that apply are defined, for example, when the table is created in the database system and are stored in the DB2 catalog, which can be subsequently queried for the information as described above. For example, the following is exemplary code for a conventional CHECK utility to read the SYSRELS table in the DB2 catalog:

DEPR:
If no referential integrity constraints are defined for the table, then a SQL code of +100 is returned to the CHECK utility by the DB2 catalog. If a referential integrity constraint does apply to the table, however, then the DB2 catalog returns the row value for each referential integrity constraint parent table (e.g., the DB2 catalog provides the row of the SYSIBM.SYSRELS table having a column with the name of the parent table for the referential integrity constraint). More than one row can be returned by the DB2 catalog if more than one referential integrity constraint is defined for the table. The CHECK utility would then, for example, read the DB2 catalog to identify the column names for the foreign keys for the referential integrity constraints, for example by reading the SYSFOREIGNKEYS table in the DB2 catalog. Exemplary code for how a conventional CHECK utility would read the SYSFOREIGNKEYS table is set forth below:

DEPR:
If referential integrity constraints apply, the parent index for each applicable referential integrity constraint is identified by the CHECK utility. For example, the CHECK utility can read the DB2 catalog (e.g., the SYSIBM.SYSINDEX table of the DB2 catalog) for each parent table and the DB2 catalog will return the name of the parent index for the parent table. For example, the following is exemplary code for a CHECK utility to read the SYSINDEX table of the DB2

DEPR:
In addition to the CHECK utility accessing the DB2 catalog during initialization of the CHECK utility, for example to determine if any check constraints or referential integrity constraints apply to the table, as illustrated in steps 510 to 550 of FIG. 5, the CHECK utility performs additional steps 560 and 570 during initialization in an embodiment of the present invention. As shown in FIG. 5, in step 560 the CHECK utility determines the index or indexes that have been defined for the table. The CHECK utility can determine the index or indexes by reading the DB2 catalog, for example using the SYSIBM.SYSINDEXES table in the DB2 catalog. The following is exemplary code for the CHECK utility to read the SYSINDEXES table:

DEPR:
In response to this query from the CHECK utility, the DB2 catalog will return all of the indexes that have been defined for the table. For example, there will be one row in the SYSINDEXES table for each index of the table. Using the database table illustrated in FIG. 3, DB2 would return to the CHECK utility a row from the DB2 catalog indicating that the order_number index 140 illustrated in FIG. 2 has been defined for the Order_Entry table 120.

DEPR:
Also as shown in FIG. 5, in step 570, the CHECK utility identifies the column names for the index or indexes identifies in step 560. The CHECK utility can determine the column names by, for example, reading the SYSIBM.SYSKEYS table in the DB2 catalog. The following is exemplary code for the CHECK utility to read the SYSKEYS table:

DEPR:
Thus, using the Order_Entry table 120 example, step 570 returns to the CHECK utility the column name order_number as the column name of the index defined for database 120. As will be described below, this column name can be used by the CHECK utility to select the index value from a row containing a constraint violation to be used in generating a DELETE statement according to an embodiment of the present invention.

DEPR:

The generation of the DELETE [redacted]atement according to an embodim[redacted] of the present invention is facilitated, for example, by the steps performed by the CHECK utility during initialization as described with regard to FIG. 5. For example, while conventional CHECK utilities generally perform the functions illustrated in steps 510-550 of FIG. 5, according to the present invention additional steps 560 and 570 are performed so that the CHECK utility identifies the index or indexes defined for the table to be checked as well as the column names for the index or indexes.

DEPR:
For example, when the CHECK utility reads the database table 120 to perform constraint enforcement, the values contained in each column of the row are available to the CHECK utility. The column containing the index value is also known to the CHECK utility via the initialization process (e.g., steps 560 and 570 in FIG. 5) and thus if a constraint violation is identified, then the index value can be located in the appropriate column by the CHECK utility and used for generating the DELETE statement.

DEPR:
Therefore, as a result of the method according to the present invention, when the user of the database system determines that a SQL DELETE statement is to be executed, DB2 does not have to read the entire database table to identify any rows containing the key value that failed the check constraint. Rather, DB2 can use the column name and index value used in the DELETE statement to identify, for example, the page containing the row to be deleted and then read only that page to find and delete the row containing the constraint violation. For example, the DELETE statements generated according to an embodiment of the present invention would utilize the column name order_number and the index values contained in column 313 of FIG. 3, which would allow DB2, when executing the DELETE statement, to access the index 140 to identify the RID of the row to be deleted. Accordingly, by generating the DELETE statements in the manner according to an embodiment of the present invention, the need to read millions of rows in a database table has been eliminated and only the rows on the page containing the row to be deleted are read.

DEPR:
FIG. 7 illustrates exemplary error processing according to an embodiment of the present invention when, for example, a referential integrity constraint violation is identified in steps 460 and 470 of FIG. 4. In step 710 of FIG. 7, an error report is generated. The error report can include, for example, a printout of the row identification and value violating the referential integrity constraint, similar to the error report for a check constraint violation. In step 720, the CHECK utility determines if a DELETE statement has already been generated for the row containing the referential integrity constraint violation, for example in the manner described with regard to FIG. 6. If step 720 determines that no DELETE statement has been generated for the row, then a DELETE statement is generated for the row in step 740 using the column name and index value associated with the row and not the particular key value that violated the referential integrity constraint, in the same manner described above with respect to FIG. 6. If a DELETE statement has already been generated, then the CHECK utility returns to the main flow in step 730, for example reading the next row in the database table.

DEPR:
If, for example, the index defined for a database table was not based on a unique value, then the key value that violates the constraint can be used in conjunction with the non-unique index value. For example, if an index was defined for database table 120 based on ship_to_zip, the index value would not uniquely identify a particular row in the table but rather would identify a subset of rows in the table (e.g., all the rows on the table having a particular ship_to_zip value). Thus, a key value violating a constraint, such as customer number=99999, could be combined with the index value for use in generating the SQL DELETE statement in the CHECK utility. For example, the DELETE statement could have the form of:

DEPR:
In addition to the generation of SQL DELETE statements based on an index value, an output file is also generated for the database table including a SQL UPDATE statement for each row containing a constraint violation. Thus, according to an embodiment of the present invention, a user of the database table has the option of deleting a row in error or correcting the row using the UPDATE statement.

DEPR:
FIG. 8 illustrates an exemplary output file 800 generated according to an embodiment of the present invention to facilitate repair of a constraint violation in a row of a database table. The output file 800 can be generated, for example, via a conventional CHECK utility or by a separate stand-alone utility for repairing constraint violations according to an embodiment of the present invention. Regardless of the particular implementation of the method for repairing constraint violations according to the present invention, the information needed to carry out the invention can be obtained,

for example, by reading the [redacted] catalog during an initializati[redacted] phase as described above with regard to FIG. 5. For example, following step 570 in FIG. 5, additional step 580, indicated in dashes, can be performed to generate the first and second portions of output file 800 based on the information obtained from steps 510-570. The format illustrated in

DEPR:
FIG. 8 illustrates a first portion 810 of the output file 800 including, for example, the name of the table, the creator of the table, the OBID of the table, the rowsize of the table, whether any referential integrity or check constraints apply to the table, the subsystem ID of the table, the database name of the table, the tablespace name for the table and values for EDITPROC, VALIDPROC, AUDIT, RESTRICT and EXPLAIN for the table. The information provided in section 810 can be obtained by, for example, reading the DB2 catalog and provides a user of the database table with details on the structure of the database table.

DEPR:
Section 820 of output file 800 provides, for example, the column number and name for each column in the database table along with the column type and size of each column. For example, the column type can be character, data, integer, time, variable character, decimal, etc. The information provided in section 820 can be obtained by, for example, reading the DB2 catalog. By providing a user of the database table with the information contained in sections 810 and 820 of output file 800, knowledge of the structure and content of the database table can be provided to the user at the time corrective action is needed to repair a row. If the user decides to correct a row of the database table, the user has sufficient information to carry out such a task, for example the column name, column type and byte size for particular columns which are needed to take corrective action for a row in error.

DEPR:
According to an embodiment of the present invention, a user does not need to generate corrective SQL statements as required with conventional database systems. SQL UPDATE statements 830 are automatically generated as shown in FIG. 8 as part of the output file 800. The utility implementing the method for repairing constraint violations according to an embodiment of the present invention can be coded in software in any conventional manner to generate the exemplary format for an SQL UPDATE statement illustrated as element 830 in FIG. 8. The generation of a SQL UPDATE statement 830 can be triggered, for example, by the identification of a constraint violation if the repair method is included in a CHECK UTILITY. For example, FIGS. 6 and 7 illustrate error processing according to an embodiment of the present invention including additional steps 650 and 750, respectively, for generating a SQL UPDATE statement 830 for each row containing a constraint violation after a SQL DELETE statement is generated for the row. Using the information obtained by the CHECK utility in performing constraint enforcement, the constraint repair utility would generate the SQL UPDATE statement and write the statement into output file 800.

DEPR:
The information provided in sections 810 and 820 of output file 800 enhance the user's ability to revise the contents of the UPDATE statement 830 to repair a constraint violation in a row of a database table according to an embodiment of the present invention. Each UPDATE statement 830 generated in response to a constraint violation includes, for example, the SQL UPDATE command for the database table to be updated as well as a listing of the value contained in each column of the row, identified as elements 831a-831i or a subset of the values identified as elements 831a-831i. In another embodiment of the present invention, the SQL UPDATE statement 830 can include only the column value causing the constraint violation. If a user reviewed , for example, the first UPDATE statement 830 and wanted to change the value 831g in the comment column, the information provided in section 820 would inform the user that any value placed in the comment column could have a size of up to 25 bytes. As the user may be responsible for or interact with numerous tables, each of which have a different structure, providing the details on the configuration of each database table in sections 810 and 820 as well as the row in error greatly facilitates the user's ability to effectively and efficiently repair constraint violations.

DEPR:
According to an embodiment of the present invention, all that a user must do to repair a constraint violation in a row of a database table is revise the data provided in section 830, which is the SQL UPDATE statement to be used to implement the repair and which has already been generated in response to the identification of the constraint violation. Once revised, the SQL UPDATE statement shown as section 830 can be executed by DB2 to repair the constraint violation--the user does not need to generate or debug any SQL statements. The benefit to a user trying to correct rows including a constraint violation according to the present invention utilizing for example, a pre-generated SQL UPDATE statement including a current set of values for each column of the row that can

be modified as well as information on the structure and content of the database table is exemplified when compared to the provision of only the RID and constraint violation by prior art CHECK utilities, which are of limited value in helping a user identify the data in the row and generate the SQL statement needed to fix the row.

DEPR:
Without the output file according to an embodiment of the present invention, in order to repair rows of a database table containing a constraint violation, a user would have to obtain the necessary values for the row in error to generate an SQL UPDATE statement for each row. While the error report provided by a conventional CHECK UTILITY only identifies the name of the row in error and thus does not implement any corrective action other than deletion, the error report can be used in conjunction with the output file 800 according to an embodiment of the present invention to repair a constraint violation. For example, the RID listed in an error report can be used for correlation to the SQL UPDATE statement 830 for the same RID, identified in statement 830 as element 831i.

DEPR:
According to the present invention, a stand-alone utility or a CHECK utility incorporating the method according to the present invention can generate the output file 800. Once the output file is created, the customer updates the output file 800 as necessary and then executes the output file to repair the constraint violations. Thus, the user no longer has to manually generate a SQL statement for each row with an error. The output file 800 is executed using, for example, SPUFI, the output file 800 according to an embodiment of the present invention being the input file to SPUFI. The executed output file can operate on, for example, the database table to be corrected or an exception table as described below.

DEPR:
In another embodiment of the method for repairing constraint violations according to the present invention, an exception table is used. As is known in the art, an exception table is generated prior to each time a CHECK utility operates upon a table (e.g., a new exception table is generated or a prior exception table replaced each time constraint enforcement is performed). For example, when a user creates a job stream to execute a CHECK utility, a step of the job stream includes creating a new exception table. The exception table is, for example, a mirror image of the database table except that the exception table only contains the rows including a constraint violation. For example, each time a CHECK utility identifies a constraint violation, the CHECK utility copies the entire row into the exception table. Exemplary code to copy rows in error into an exception table is as follows.

DEPR:
When the SQL UPDATE statements 830 are generated according to an embodiment of the present invention and a user has made the desired corrections to the UPDATE statements, then the UPDATE statements 830 are executed and operate upon the rows stored in the exception table. Applying the UPDATE statements against the rows in the exception table minimizes the number of changes applied to the database table and provides an opportunity for the user to verify that the proper corrections have been made to each row. Once the user is satisfied that the proper corrections have been made to the rows in the exception table, then the corrected rows can be inserted into the database table. For example, a SQL INSERT statement can be used to insert the corrected rows from the exception table into the database table. A sample SQL INSERT statement is as follows.

DEPL:
If no check constraints are defined for the table, then, for example, DB2 returns to the CHECK utility a SQL code of +100. If a check constraint is defined for the table, then the DB2 catalog returns a row value identifying the check constraint. The row value is the check predicate from which check constraint routines can be compiled and built by the CHECK utility, as is known in the art.

DEPL:
Therefore, utilizing the method for enforcing constraints according to the present invention with a database table having a defined index based on non-unique values, the number of rows to be read can be significantly reduced by generating SQL DELETE statements for rows containing constraint violations utilizing an index value associated with the row. For example, the above example would result in DB2 reading the page containing each row having a ship_to_zip value of 60606 to determine if any such row contained a customer_number of 99999. In contrast, without the method according to the present invention, each row of the database table would have to be read when the DELETE statement was executed instead of only a subset of rows.

DEPL:
As shown by the above code, a row containing a constraint violation in database table PDLNR.TDOCDPP will be copied into exception table PDLNR.EXTDOCDPP4.

DEPL:
Using, for example, the above code, the values contained in the various columns (e.g., DEPT, COMMENT, PRICE) for each row that has been repaired are inserted into the database table PDLNR.TDOCDPP4 from the exception table PDLNR.EXTDOCDPP4. As mentioned previously, the UPDATE and INSERT statements can also utilize only the value causing the constraint violation. The dashes before each line of code indicates a comment line which is not executed by the utility when the UPDATE statements are generated and executed. Once the UPDATE statements have been satisfactorily applied to the exception table, then the dashes can be removed and the SQL INSERT statement executed by DB2 to move the repaired rows into the database table.

CLPR:
1. A method for repairing a constraint violation in a database table, comprising the steps of:

CLPR:
2. The method according to claim 1, wherein the output file includes a description of the database table.

CLPR:
3. The method according to claim 2, wherein the description includes a name of each column in the database table.

CLPR:
4. The method according to claim 3, wherein the description further includes a column type for each column of the database table.

CLPR:
5. The method according to claim 4, wherein the description further includes a size of each column in the database table.

CLPR:
6. The method according to claim 1, wherein the update statement includes a SQL UPDATE statement operable with a DB2 database management system.

CLPR:
11. The method according to claim 1, further comprising the step of executing a CHECK utility on the database table to identify the row including the constraint violation.

CLPR:
13. A method for repairing a constraint violation in a database table, comprising the steps of:

CLPR:
17. The method according to claim 16, wherein the step of replacing the row includes inserting the repaired row from an exception table into the database table.

CLPR:
18. The method according to claim 13, wherein the step of generating the output file includes storing information describing a configuration of the database table in the output file.

CLPV:
generating an output file for a database table, the database table containing a row including a constraint violation; and

CLPV:
inserting the repaired row into the database table.

CLPV:
inserting the repaired row from an exception table into the database table.

CLPV:
generating an output file for a database table containing a row having a constraint violation; and

CLPV:
replacing the row in the database table containing the constraint violation with the repaired row.

ORPL:
Yoon, J.P., et al., "Semantic update optimization in active databases", Database Applications Semantics, Proc. of the IFIP WG 2.6 Working Conf. on DB Appl Semantics,

Jun. 1995, pp. 1-26.

ORPL:
Yoon, J.P. et al., "Databases updates using active rules: a unified approach for consistency maintenance", DB Systems for Adv Applications, '93 Proc. 3d Int'l Symp. on DB, 1993, pp. 271-278.

Generate Collection

L3: Entry 41 of 315 File: USPT Feb 13, 2001

US-PAT-NO: 6189010
DOCUMENT-IDENTIFIER: US 6189010 B1

TITLE: Method for repairing constraint violations in a database management system

DATE-ISSUED: February 13, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Isip, Jr.; Amando B.	Richardson	TX	N/A	N/A

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Platinum Technology, Inc.	Oakbrook Terrace	IL	N/A	N/A	02

APPL-NO: 9/ 095449
DATE FILED: June 10, 1998

INT-CL: [7] G06F 17/30
US-CL-ISSUED: 707/100; 707/8
US-CL-CURRENT: 707/100; 707/8
FIELD-OF-SEARCH: 707/3, 707/9, 707/102, 707/100, 707/2, 707/8

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4933848	June 1990	Haderle et al.	364/300
☐	4947320	August 1990	Crus et al.	364/200
☐	5226158	July 1993	Horn et al.	395/600
☐	5241648	August 1993	Cheung et al.	395/600
☐	5386557	January 1995	Boykin et al.	707/1
☐	5513350	April 1996	Griffin et al.	395/702
☐	5551029	August 1996	Jagadish et al.	707/103
☐	5553218	September 1996	Li et al.	707/102
☐	5706494	January 1998	Cochrane et al.	707/2
☐	5745896	April 1998	Vijaykumar	707/100
☐	5873075	February 1999	Cochrane et al.	707/2
☐	5899993	May 1999	Jenkins, Jr.	707/9
☐	5950188	September 1999	Wildermuth	707/3
☐	5950210	September 1999	Nelson	707/203
☐	6065017	May 2000	Barker	707/202

OTHER PUBLICATIONS

Yoon, J.P., et al., "Semantic update optimization in active databases", Database Applications Semantics, Proc. of the IFIP WG 2.6 Working Conf. on DB Appl Semantics, Jun. 1995, pp. 1-26.
Baralis, E. et al., "Declarative specification of constraint maintenance", Entity-Relationship Approach--ER '94, 13th Int'l Conf. on ER Approach Proc., Dec. 1994, pp. 205-222.
Yoon, J.P. et al., "Databases updates using active rules: a unified approach for consistency maintenance", DB Systems for Adv Applications, '93 Proc. 3d Int'l Symp. on DB, 1993, pp. 271-278.

ART-UNIT: 277
PRIMARY-EXAMINER: Homere; Jean R.
ATTY-AGENT-FIRM: Baker & McKenzie

ABSTRACT:

In response to a constraint violation in a row of a database table, an output file is generated including the characteristics of the table containing the row in error as well as an SQL UPDATE statement for the row. The SQL UPDATE statement includes the column values in the row which can be corrected by the user, the user modified SQL UPDATE statement being subsequently executed to repair the constraint violation.

18 Claims, 8 Drawing figures

Generate Collection

L10: Entry 4 of 21 File: USPT Sep 12, 2000

US-PAT-NO: 6119128
DOCUMENT-IDENTIFIER: US 6119128 A

TITLE: Recovering different types of objects with one pass of the log

DATE-ISSUED: September 12, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Courter; Daniel Keith	Antioch	CA	N/A	N/A
Hu; Ming-Hung	San Jose	CA	N/A	N/A
Kunioka-Weis; Laura Michiko	Morgan Hill	CA	N/A	N/A
Majithia; Thomas	San Jose	CA	N/A	N/A
Matamoros; Deborah A.	San Jose	CA	N/A	N/A
Ruddy; James Alan	Gilroy	CA	N/A	N/A
Wang; Yufen	Saratoga	CA	N/A	N/A

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY	N/A	N/A	02

APPL-NO: 9/ 050554
DATE FILED: March 30, 1998

INT-CL: [7] G06F 12/00
US-CL-ISSUED: 707/202; 707/200, 707/201, 707/203, 707/204, 707/205
US-CL-CURRENT: 707/202; 707/200, 707/201, 707/203, 707/204, 707/205
FIELD-OF-SEARCH: 707/200, 707/201, 707/202, 707/203, 707/204, 707/205

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4945474	July 1990	Elliott et al.	714/16
☐	5276872	January 1994	Lomet et al.	707/202
☐	5278982	January 1994	Daniels et al.	707/202
☐	5280611	January 1994	Mohan et al.	707/8
☐	5327532	July 1994	Ainsworth et al.	709/248
☐	5333303	July 1994	Mohan	714/20
☐	5455944	October 1995	Haderle et al.	707/202
☐	5455946	October 1995	Mohan et al.	707/202
☐	5561795	October 1996	Sarkar	707/202
☐	5561798	October 1996	Haderle et al.	707/202
☐	5574897	November 1996	Hermsmeier et al.	707/1
☐	5581750	December 1996	Haderle et al.	707/202
☐	5625820	April 1997	Hermsmeier et al.	707/202
☐	5721918	February 1998	Nilsoson	707/202
☐	5832508	November 1998	Sheman	707/200
☐	5873096	February 1999	Lim	707/201
☐	5903898	May 1999	Cohen	707/204
☐	5907848	May 1999	Zaiken	707/202
☐	5920873	July 1999	Van Huben	707/202
☐	5926816	July 1999	Bauer	707/8

OTHER PUBLICATIONS

"Incremental Data Base Log Image Copy", IBM Technical Disclosure Bulletin, vol. 25, No. 7B, pp. 3730-3732, 1982.
"Technique For Data Recovery excluding Portions of The Log Without Requiring a Full Image Copy", IBM Technical Disclosure Bulletin, vol. 36, No. 11, pp. 359-360, Nov. 1993.
Fernando de Ferreira Rezende, et al., "Employing Object-Based LSNs in A Recovery Strategy", Database And Expert Systems Applications, 7th International Conference, DEXA '96, pp. 116-129, Sep. 9-13, 1996.

ART-UNIT: 271
PRIMARY-EXAMINER: Black; Thomas G.
ASSISTANT-EXAMINER: Mizrahi; Diane D.
ATTY-AGENT-FIRM: Pretty, Schroeder & Poplawski

ABSTRACT:

A method, apparatus, and article of manufacture for a computer implemented recovery system for restoring a database in a computer. The database contains objects and is stored on a primary data storage device connected to the computer. Objects of different types in the database are copied from the primary data storage device to a secondary data storage device. Modifications to the objects are logged in a log file. A recovery indicator is received that indicates that recovery of the objects in the database is required. The objects are copied from the secondary data storage device to the database on the primary data storage device. Modifications in the log file are applied to the copied objects during one pass through the log file.

24 Claims, 6 Drawing figures

L10: Entry 4 of 21 File: USPT Sep 12, 2000

DOCUMENT-IDENTIFIER: US 6119128 A
TITLE: Recovering different types of objects with one pass of the log

ABPL:
A method, apparatus, and article of manufacture for a computer implemented recovery system for restoring a database in a computer. The database contains objects and is stored on a primary data storage device connected to the computer. Objects of different types in the database are copied from the primary data storage device to a secondary data storage device. Modifications to the objects are logged in a log file. A recovery indicator is received that indicates that recovery of the objects in the database is required. The objects are copied from the secondary data storage device to the database on the primary data storage device. Modifications in the log file are applied to the copied objects during one pass through the log file.

BSPR:
This invention relates in general to computer-implemented database systems, and, in particular, to recovering different types of objects with one pass of the log.

BSPR:
Databases are computerized information storage and retrieval systems. A Relational Database Management System (RDBMS) is a database management system (DBMS) which uses relational techniques for storing and retrieving data. Relational databases are organized into tables which consist of rows and columns of data. The rows are formally called tuples. A database will typically have many tables and each table will typically have multiple tuples and multiple columns. The tables are typically stored on direct access storage devices (DASD), such as magnetic or optical disk drives for semipermanent storage.

BSPR:
A table is assigned to a tablespace. The tablespace contains one or more datasets. In this way, the data from a table is assigned to physical storage on DASD. Each tablespace is physically divided into equal units called pages. The size of the tablespace's pages is based on the page size of the bufferpool specified in the tablespace's creation statement. The bufferpool is an area of virtual storage that is used to store data temporarily. A tablespace can be partitioned, in which case a table may be divided among the tablespace's partitions, with each partition stored as a separate dataset. Partitions are typically used for very large tables.

BSPR:
A table may have an index. An index is an ordered set of pointers to the data in the table. There is one physical order to the rows in a table that is determined by the RDBMS software, and not by a user. Therefore, it may be difficult to locate a particular row in a table by scanning the table. A user creates an index on a table, and the index is based on one or more columns of the table. A partitioned table must have at least one index. The index is called the partitioning index and is used to define the scope of each partition and thereby assign rows of the table to their respective partitions. The partitioning indexes are created in addition to, rather than in place of, a table index. An index may be created as UNIQUE so that two rows can not be inserted into a table if doing so would result in two of the same index values. Also, an index may be created as a CLUSTERING index, in which case the index physically stores the rows in order according to the values in the columns specified as the clustering index (i.e., ascending or descending, as specified by the user).

BSPR:
RDBMS software using a Structured Query Language (SQL) interface is well known in the art. The SQL interface has evolved into a standard language for RDBMS software and has been adopted as such by both the American National Standards Institute (ANSI) and the International Standards Organization (ISO). The SQL interface allows users to formulate relational operations on the tables either interactively, in batch files, or embedded in host languages, such as C and COBOL. SQL allows the user to manipulate the data. As the data is being modified, all operations on the data are logged in a log file.

BSPR:
Typically, the database containing partitions and indexes is stored on a data storage device, called a primary data storage device. The partitions are periodically copied to another data storage device, called a secondary data storage device, for recovery purposes. In particular, the partitions stored on the primary data storage device may be corrupted, for example, due to a system failure during a flood, or a user may want to remove modifications to the data (i.e., back out the changes). In either case, for recovery, the partitions are typically copied from the secondary data storage device to the primary data storage device. Next, using the log file, the copied data is modified based on the operations in the log file. Then, the indexes are rebuilt. In particular, to rebuild the indexes, keys are copied from each row of each partition, sorted, and then used to create a partitioning index. Additionally, the table index is rebuilt via the same technique.

BSPR:
To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for a computer implemented recovery system for restoring a database in a computer.

BSPR:
In accordance with the present invention, the database contains objects and is stored on a primary data storage device connected to the computer. Objects of different types in the database are copied from the primary data storage device to a secondary data storage device. Modifications to the objects are logged in a log file. A recovery indicator is received that indicates that recovery of the objects in the database is required. The objects are copied from the secondary data storage device to the database on the primary data storage device. Modifications in the log file are applied to the copied objects during one pass through the log file.

BSPR:
An object of the invention is to provide an improved recovery system for a database. Another object of the invention is to provide recovery for partitions, partitioning indexes, and table indexes simultaneously. Yet another object of the invention is to provide a recovery system for a database that requires only one pass of a log file to apply modifications to the database.

DRPR:
FIG. 2 illustrates a conventional system for recovery of a database;

DRPR:
FIG. 5 is a flow diagram illustrating the steps performed by the recovery system prior to recovery of a database in accordance with the present invention; and

DRPR:
FIG. 6 is a flow diagram illustrating the steps performed by the recovery system to recover a database in accordance with the present invention.

DEPR:
FIG. 1 illustrates an exemplary computer hardware environment that could be used in accordance with the present invention. In the exemplary environment, a computer system 102 is comprised of one or more processors connected to one or more data storage devices 104 and 106 that store one or more relational databases, such as a fixed or hard disk drive, a floppy disk drive, a CDROM drive, a tape drive, or other device.

DEPR:
Operators of the computer system 102 use a standard operator interface 108, such as IMS/DB/DC.RTM., CICS.RTM., TSO.RTM., OS/390.RTM. or other similar interface, to transmit electrical signals to and from the computer system 102 that represent commands for performing various search and retrieval functions, termed queries, against the databases. In the present invention, these queries conform to the Structured Query Language (SQL) standard, and invoke functions performed by Relational DataBase Management System (RDBMS) software. In the preferred embodiment of the present invention, the RDBMS software comprises the DB2.RTM. product offered by IBM for the MVS.RTM. or OS/390.RTM. operating systems. Those skilled in the art will recognize, however, that the present invention has application program to any RDBMS software that uses SQL.

DEPR:
As illustrated in FIG. 1, the DB2.RTM. architecture for the MVS.RTM. operating system includes three major components: the Internal Resource Lock Manager (IRLM) 110, the Systems Services module 112, and the Database Services module 1 14. The IRLM 110 handles

locking services for the DB[redacted]TM. architecture, which treats d[redacted]a as a shared resource, thereby allowing any number of users to access the same data simultaneously. Thus concurrency control is required to isolate users and to maintain data integrity. The Systems Services module 112 controls the overall DB2.RTM. execution environment, including managing log data sets 106, gathering statistics, handling startup and shutdown, and providing management support.

DEPR:
At the center of the DB2.RTM. architecture is the Database Services module 114. The Database Services module 114 contains several submodules, including the Relational Database System (RDS) 116, the Data Manager 118, the Buffer Manager 120, the Recovery System 122, and other components 124, such as an SQL compiler/interpreter. These submodules support the functions of the SQL language, i.e. definition, access control, interpretation, compilation, database retrieval, and update of user and system data. The Recovery System 122 works with the components of the computer system 102 to restore a database.

DEPR:
The present invention is generally implemented using SQL statements executed under the control of the Database Services module 114. The Database Services module 114 retrieves or receives the SQL statements, wherein the SQL statements are generally stored in a text file on the data storage devices 104 and 106 or are interactively entered into the computer system 102 by an operator sitting at a monitor 124 via operator interface 108. The Database Services module 114 then derives or synthesizes instructions from the SQL statements for execution by the computer system 102.

DEPR:
The present invention provides a recovery system 122 for recovering different types of objects using only one pass through a log file. In particular, table partitions of a database, along with indexes (e.g., partitioning indexes and table indexes), are copied to one or more data storage devices, such as magnetic tape. The database may be stored on a primary data storage device, while the copies of the database partitions and indexes are stored on a secondary data storage device. The primary and secondary data storage devices could be the same or different devices.

DEPR:
Then, as modifications are made to the data in the table partitions, the

DEPR:
modifications are logged in a log file. If recovery of the table partitions and partitioning indexes are required, the recovery system 122 of the present invention copies the table partitions and partitioning indexes from the secondary data storage device back to the database. Then, the recovery system 122 modifies both the table partitions and the partitioning indexes while making one pass through the log file. That is, the recovery system 122 extracts all of the pertinent log records containing updates to all of the objects being recovered in a single read pass of logged changes.

DEPR:
The recovery system 122 allows for independent recovery of the data and indexes, and a significant decrease in elapsed time since the log file updates are done for all objects in the database with one pass through the log file.

DEPR:
FIG. 2 illustrates a conventional system for recovery of a database. In a conventional system, partitions 200 and 202 of a database are copied from primary data storage devices to secondary data storage devices 204 and 206. In a conventional system, the partitioning indexes 208 and 210 and the table index 212 are not stored on secondary data storage devices. Then, when recovery is required, the conventional system copies the partitions 200 and 202 from the secondary data storage devices 204 and 206 to the database on the primary data storage devices. The conventional system applies modifications logged in a log file to the copied partitions. Then, the conventional system reads each row of each partition 200 and 202 and retrieves index keys 214 and 216 for each row of each partition 200 and 202. The index keys 214 and 216 are sorted and are used to rebuild indexes 208 and 210, respectively. Table index 212 is rebuilt in the same manner. This procedure has a high performance cost.

DEPR:
FIG. 3 illustrates the recovery system 122 in accordance with the present invention. Initially, the partitions 300 and 302 are copied to secondary data storage devices 304 and 306. Also, partitioning indexes 308 and 312 are copied to secondary data storage devices 310 and 314. The table index 316 is also copied to a secondary data storage device 318. Then, as application programs 320 modify the database by adding, updating, or deleting data via operations, the modifications are logged in the log file 322. The log file may be copied to a secondary data storage device 324 if the log file on the

primary storage device becomes full. The log file 322 contains information identifying modifications to both the partitions and indexes.

DEPR:
For recovery, the partitions from the data storage devices 304 and 306 are copied back to the primary data storage device. The partitioning indexes are copied from the secondary data storage devices 310 and 314 to the primary data storage device. Additionally, the table index is copied from the secondary data storage device 318 to the primary data storage device. Then, the log records subsequent to the last copying from the primary to the secondary data storage devices are applied to the partitions and indexes. In particular, while reading the log file through once, the recovery system 122 modifies both the partitions 300 and 302 and the indexes 308, 312, and 316.

DEPR:
Next, if the name of an employee is changed, the partition 400 and partitioning index 404 are modified. Then, the log file 410 contains an entry for "New Name, Old Name" that provides the new and old name of the employee whose name changed and an entry for "New Name Index, Old Name Index" that provides the index modification. Next, assuming that there is a loss of data, recovery of the data is required. Initially, the partition 400 and the partitioning index 404 are copied from secondary data storage devices back to the primary data storage device. Since, according to the copies file 416, these copies include all modifications up to range identifier L2, only operations after range identifier L2 are applied to the partition 400 and the partitioning index 404 to recover the database. Moreover, during one pass through the log file, the recovery system 122 identifies the required modifications and applies them to the partition 400 and the partitioning index 404.

DEPR:
FIG. 5 is a flow diagram illustrating the steps performed by the recovery system 122 prior to recovery of a database in accordance with the present invention. In Block 500, the recovery system 122 copies objects from a primary data storage device to a secondary data storage device. In Block 502, the recovery system 122 logs all operations for each object in the database that is modified. In Block 504, the recovery system 122 receives a recovery indicator.

DEPR:
FIG. 6 is a flow diagram illustrating the steps performed by the recovery system 122 to recover a database in accordance with the present invention. In Block 600, the recovery system 122 copies objects from the secondary data storage device to the primary data storage device. That is, each of the objects is replaced by an image copy taken at a previous time. The individual objects may be restored from the image copies concurrently with each other. In Block 602, the recovery system 122 determines the point in the log at which to start applying operations. In Block 604, the recovery system 122 applies log operations to all objects through one pass of the log file.

DEPR:
In summary, the present invention discloses a method, apparatus, and article of manufacture for a computer-implemented recovery system. The present invention provides an improved recovery system for a database. Additionally, the present invention provides recovery for partitions and partitioning indexes simultaneously. Moreover, the present invention provides a recovery system for a database that requires only one pass of a log file to apply modifications to the database.

CLPR:
1. A method of restoring a database in a computer, the database containing objects and being stored on a primary data storage device connected to the computer, the method comprising the steps of:

CLPR:
2. The method of claim 1, wherein the types of the objects include table data.

CLPR:
9. An apparatus for restoring a database in a computer, comprising:

CLPR:
10. The apparatus of claim 9, wherein the types of the objects include table data.

CLPR:
17. An article of manufacture comprising a computer program carrier readable by a computer and embodying one or more instructions executable by the computer to perform method steps for restoring a database, the database containing objects and being stored on a primary data storage device connected to the computer, the method comprising the steps of:

CLPR:
18. The method of claim 17, wherein the types of the objects include table data.

CLPV:
copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition;

CLPV:
logging modifications to the objects, including the table index and the partitioning index, in a log file;

CLPV:
receiving a recovery indicator indicating that recovery of the objects in the database is required;

CLPV:
copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device; and

CLPV:
applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

CLPV:
a computer having a primary data storage device connected thereto, wherein the primary data storage device stores a database containing objects;

CLPV:
one or more computer programs, performed by the computer, for copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition, logging modifications to the objects, including the table index and the partitioning index, in a log file, receiving a recovery indicator indicating that recovery of the objects in the database is required, copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device, and applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

CLPV:
copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition;

CLPV:
logging modifications to the objects, including the table index and the partitioning index, in a log file;

CLPV:
receiving a recovery indicator indicating that recovery of the objects in the database is required;

CLPV:
copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device; and

CLPV:
applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

ORPL:
Fernando de Ferreira Rezende, et al., "Employing Object-Based LSNs in A Recovery Strategy", Database And Expert Systems Applications, 7th International Conference, DEXA '96, pp. 116-129, Sep. 9-13, 1996.

L6: Entry 23 of 26 File: USPT Jul 4, 1995

DOCUMENT-IDENTIFIER: US 5430871 A
TITLE: Method of dynamically adding and removing DB2 active logs

BSPR:
This invention relates to the use of database software such as the well-known DATABASE 2 database software distributed by IBM Corporation, commonly referred to colloquially in the art as "DB2." As is well known to those of ordinary skill, generally speaking DB2 operates as a subsystem in a computer system that itself is operating under the IBM MVS operating system software. More specifically, the invention relates to a method by which a user of a DB2 application can dynamically add and remove DB2 active logs without affecting the availability of the database system or of DB2 applications.

BSPR:
DB2 utilizes a fixed number of active logs, with the number and size of the logs being set at DB2 initialization time. DB2 may be configured by the database administrator (DBA) to maintain two or more active logs in memory, of which one is always the "current active log."

BSPR:
As illustrated in simplified form in FIG. 3, DB2 maintains the active logs as, in effect, a single logically infinite log by performing a round-robin type of operation, switching from a full active log (e.g., ACT3 if FIG. 3) to the least recently used active log data set (e.g., ACT1) and overwriting it. This normally does not cause a problem because the active-log information in the least recently used active log data set is ordinarily copied off to an archive log prior to being selected for overwriting. At the database administrator's option, the DB2 subsystem can be operated in "dual active log chain" mode. In that mode, for redundancy purposes the active log chain is kept in duplicate copies, referred to as copy 1 and copy 2.

BSPR:
The BSDS includes an active log copy 1 record; if the DB2 subsystem was started in dual active log mode the BSDS also includes an active log copy 2 record. Each of the active log copy 1 and 2 records comprise a 4K page of storage, which effectively limits the size of the active log chain because when the available space in the BSDS record is exhausted, no more active logs can be added. The BSDS also includes records in which DB2 keeps track of its archive data sets. Whenever DB2adds, or archives, an active log, it updates the active log record (including the active log's relative byte address or RBA, assigned by DB2) in the BSDS.

BSPR:
In environments where computer-system and application-program availability is a prime concern, it is normally desirable to have enough active log space to perform timely recovery. When a new DB2 application is brought up, for example, it can sometimes be difficult to predict how many insertions, deletions, and updates may occur in the data associated with that application. That in turn makes it difficult to predict the net additional active log requirements. If data compression is used in a DB2 application, it can be even harder to predict active log requirements. Data compression causes each row (record) to be treated as a variable-length row; that means that any change taking place in a column (field) will be logged as well as all dam to the end of the row in question. Moreover, although the current generation of DASD devices is generally regarded as quite reliable, and although DB2 has the ability to use dual active logs for redundancy, I/O errors can still pose problems of downtime or concern about the availability of DB2.

BSPR:
Furthermore, if DB2 must recover corrupted data (e.g., if a table space becomes corrupted), the recovery process generally proceeds much faster if it utilizes an active log as opposed to an archive log. If a database administrator installs ("brings up") a new DB2 application, a situation may occur in which the amount of active log space is suboptimal, possibly resulting in degraded performance and recoverability.

BSPR:

A long-felt need has thus existed among database administrators for the capability of dynamically adjusting the active log parameters of DB2 without cycling DB2.

BSPL:
The DB2 Database System

DEPR:
One illustrative embodiment of a method in accordance with the invention is described below as it might be implemented in a computer program (referred to here for convenience as "NEWPROG"). An actual implementation of such a program might be executable on, e.g., an IBM System 370-compatible machine or equivalent running IBM OS/VS SP2 (MVS/XA), SP3 or SP4 (MVS/ESA) and IBM Database 2 (DB2) version 1.3 or later, or on one or more other suitable computer systems. For purposes of further illustration, the microfiche appendix sets out selected extracts of source code from an actual software package owned and distributed under license by the assignee under the trademark OPERTUNE.

DEPR:
When invoked by a user such as a DB2 database administrator, NEWPROG carries out the method of adding or removing active logs, as requested by the database administrator, by issuing a request to the MVS operating system that a main routine be scheduled for execution under an SRB (service request block). That main routine performs or invokes performance of the functions described below.

DEPR:
Block 6.11: Because the ACE control block in DB2 includes the EB control block, initialization of the ACE.sub.-- X control block also includes setting up an EB.sub.-- X control block, as part of the ACE.sub.-- X control block, to emulate the EB control block of DB2. The EB.sub.-- X control block is initialized by writing to that control block (i) the length of the EB.sub.-- X, as determined above, (ii) the DB2 identifying code of the EB, (iii) a pointer to the DB2 control block RMVT, (iv) the EB flag A, (v) the EB flag D, (vi) the EB's home ASCE, and (vii) a flag indicating whether the routine running at any particular time is running in the DB2 master address space or in the DB2 data base's address space.

DEPR:
Referring to FIG. 7, NEWPROG checks to ensure that sufficient storage in the BSDS active log record(s) is available to add an active log; if not, an error message is returned. NEWPROG then attempts to locate a TCB in the DB2 master address space and schedules an IRB to run under that TCB and carry out essentially all of the rest of the functions described below.

DEPR:
At block 7.7, NEWPROG updates the BSDS to reflect the new active log, first obtaining DB2's BSDS ACCESS latch to keep DB2 from making unpredictable changes during this process. If DB2 is in dual-BSDS mode (determined by the database administrator at DB2 startup time), NEWPROG makes corresponding changes to both BSDSs. The required operations are, in essence, those performed by the prior-art DB2 batch utility for adding an active log when DB2 is down, i.e., adding an entry to either the active log copy 1 or copy 2 record and registering a new active log in the BSDS.

DEPR:
Address space: An MVS term used to identify a collection of tasks used to perform a set of functions. (Primary address space, secondary address space)

DEPR:
Address space control block: An MVS control block encoding information about, and used to represent, an address space.

DEPR:
ASCB: Acronym for address space control block.

DEPR:
ASCE: A DB2 control block. An ASCE exists for each address space that has a thread to DB2.

DEPR:
DB2: A common nickname or acronym for the Database 2 software system distributed by International Business Machines Corporation (IBM), extensively documented in various IBM publications such as [DB2DIAG].

DEPR:
Primary address space: The default address space in which access to data and instructions is to be performed.

DEPR:
Record: In a database, a single set of formatted data. For example, in a database of employee information, a record might constitute all information about a given employee that is kept in that database. Data in a record is commonly divided into fields. See also Field, Structure.

DEPR:
Secondary address space: An address space used by some CPU instructions to access data.

DEPU:
[DB2DIAG] IBM Database 2, Version 2, Diagnosis Guide and Reference, manual

CLPR:
1. A method of dynamically adding a new active log to a DATABASE 2 (DB2) subsystem without cycling DB2, said subsystem having a log management control block (LMB) chain that includes a log data set active log (LDSD) control block for each active log, a bootstrap data set (BSDS) data set having at least one BSDS active log record, and a cache including cached portions of the BSDS data set, said method comprising the steps of:

CLPR:
2. A method of dynamically removing a specified active log from a DATABASE 2 (DB2) subsystem without cycling DB2, said subsystem having a log management control block (LMB) chain that includes a log data set active log (LDSD) control block for each active log, a bootstrap data set (BSDS) data set, and a cache including cached portions of the BSDS data set, said method comprising the steps of: